Driver Type Clasification(Supervised)-FCD Output

In [1]:
```
import pandas as pd
```

In [2]:
```
fcd_df = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/fcd-output
```

In [3]:
```
fcd_df
```

Out[3]:

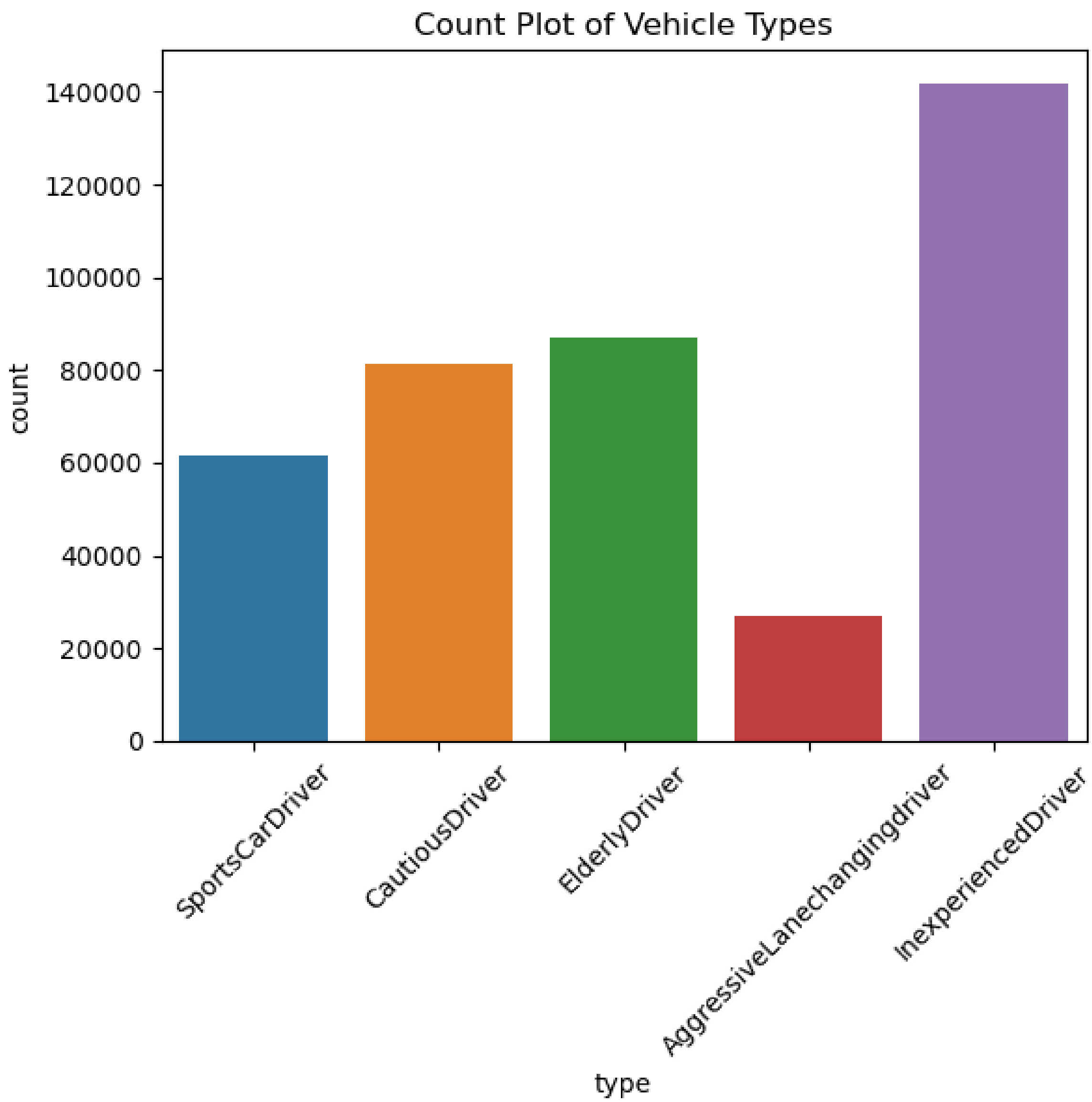
	time	id	x	y	angle	type	speed	pos	lane
0	0.0	f_0.0	-38.09	-4.80	90.00	SportsCarDriver	0.00	61.91	E0_0
1	0.0	f_1.0	-57.03	-37.67	33.37	CautiousDriver	0.00	75.68	E5_0
2	0.0	f_2.0	-33.33	-1.60	90.00	ElderlyDriver	0.00	66.67	E0_1
3	0.0	f_3.0	357.14	4.80	270.00	AggressiveLanechangingdriver	0.00	42.86	-E2_0
4	0.0	f_4.0	241.15	21.32	215.86	AggressiveLanechangingdriver	0.00	98.24	E7_0
...	...	...	...	...	...	...	...	...	...
398475	999.9	f_5.222	111.69	1.60	270.00	SportsCarDriver	0.00	102.38	-E1_2
398476	999.9	f_5.223	245.35	4.80	270.00	InexperiencedDriver	12.57	55.00	-E3_1
398477	999.9	f_5.224	158.14	4.80	270.00	SportsCarDriver	26.32	55.93	-E1_1
398478	999.9	f_5.225	335.19	1.60	270.00	ElderlyDriver	8.05	64.81	-E2_1
398479	999.9	f_5.226	367.06	4.80	270.00	ElderlyDriver	0.04	32.94	-E2_0

398480 rows × 11 columns

Count Plot of Vehicle Types

In [5]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Example of creating a count plot for the 'type' column
sns.countplot(data=fcd_df, x='type')
plt.title('Count Plot of Vehicle Types')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if need
plt.show()
```



In [6]:
```
fcd_df['id'] = fcd_df['id'].str.replace(r'^f_', '', regex=True)
```

In [7]:
```
from sklearn.preprocessing import LabelEncoder
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns
fcd_df['Type_encoded'] = label_encoder.fit_transform(fcd_df['type'])
fcd_df['lane'] = label_encoder.fit_transform(fcd_df['lane'])
```

In [8]:
```
fcd_df.drop(columns=['type'],inplace=True)
fcd_df.drop(columns=['id'],inplace=True)
```

In [9]:
```
fcd_df
```

Out[9]:

	time	x	y	angle	speed	pos	lane	slope	acceleration	Type_encoded
0	0.0	-38.09	-4.80	90.00	0.00	61.91	34	0	0.00	4
1	0.0	-57.03	-37.67	33.37	0.00	75.68	47	0	0.00	1
2	0.0	-33.33	-1.60	90.00	0.00	66.67	35	0	0.00	2
3	0.0	357.14	4.80	270.00	0.00	42.86	5	0	0.00	0
4	0.0	241.15	21.32	215.86	0.00	98.24	49	0	0.00	0
...	...	...	...	...	...	...	...	...	...	...
398475	999.9	111.69	1.60	270.00	0.00	102.38	4	0	0.00	4
398476	999.9	245.35	4.80	270.00	12.57	55.00	7	0	0.80	3
398477	999.9	158.14	4.80	270.00	26.32	55.93	3	0	-3.12	4
398478	999.9	335.19	1.60	270.00	8.05	64.81	6	0	1.15	2
398479	999.9	367.06	4.80	270.00	0.04	32.94	5	0	0.22	2

398480 rows × 10 columns

Random Forest(ensemble Method)

In [20]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Assuming 'df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = fcd_df.drop('Type_encoded', axis=1)  # Features
y = fcd_df['Type_encoded']  # Target variable

# Split the data into training and testing sets (adjust test_size and random_s
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Initialize the classifier (you can use any other classifier of your choice)
clf_rand = RandomForestClassifier(n_estimators=200, random_state=42)

# Fit the classifier on the training data
clf_rand.fit(X_train, y_train)

# Predict on the test data
y_pred = clf_rand.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9597470387472395
```

In [ ]:

In [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Assuming 'df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = fcd_df.drop('Type_encoded', axis=1)  # Features
y = fcd_df['Type_encoded']  # Target variable

# Split the data into training and testing sets (adjust test_size and random_s
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

from sklearn.model_selection import GridSearchCV

# Define the parameters for tuning
param_grid = {
    'n_estimators': [50, 100, 200]
#     'max_depth': [None, 10, 20, 30],
#     'min_samples_split': [2, 5, 10],
#     'min_samples_leaf': [1, 2, 4]
}

# Create the GridSearchCV object
grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_gri

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

print(f"Best Parameters: {best_params}")

# Use the best estimator for prediction
y_pred_tuned = best_estimator.predict(X_test)

# Calculate accuracy
accuracy_tuned = accuracy_score(y_test, y_pred_tuned)
print(f"Tuned Model Accuracy: {accuracy_tuned}")

report_data = classification_report(y_test, y_pred_tuned)
print(report_data)
```

In [ ]:

```
fcd_df
```

In [ ]:
```
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

# Assuming X_train and y_train are already defined and properly scaled
# Features (X) and target variable (y)
X = fcd_df.drop('type', axis=1)  # Features
y = fcd_df['type']  # Target variable

# Split the data into training and testing sets (adjust test_size and random_s
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Encode the labels to integers
label_encoder = LabelEncoder()
encoded_y_train = label_encoder.fit_transform(y_train)
encoded_y_test = label_encoder.transform(y_test)

# Convert integers to dummy variables (i.e., one-hot encoded)
dummy_y_train = to_categorical(encoded_y_train)
dummy_y_test = to_categorical(encoded_y_test)

# Reshape input to be [samples, time steps, features] which is required for LS
# Since your data doesn't have a time step, we can treat each row as 1 time st
X_train_reshaped = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]
X_test_reshaped = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

In [ ]:
```
X_train_reshaped.shape
```

In [ ]:
```
dummy_y_train.shape
```

In [ ]:
```
from tensorflow.keras.optimizers import Adam

# Define LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(X_train_reshaped.shape[1], X_train_reshaped.sh
model.add(Dense(dummy_y_train.shape[1], activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['ac

# Fit the model
model.fit(X_train_reshaped, dummy_y_train, epochs=100, batch_size=32, validat

# Evaluate the model
scores = model.evaluate(X_test_reshaped, dummy_y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

In [ ]:

```
# Evaluate the model on the test data
_, accuracy = model.evaluate(X_test, y_test)
print(f"Accuracy: {accuracy}")
```

DNN Model

In [11]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.utils import to_categorical

# Assuming 'fcd_df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = fcd_df.drop('Type_encoded', axis=1)  # Features
y = fcd_df['Type_encoded']  # Target variable

# Convert y to categorical (one-hot encoded)
y = to_categorical(y)
num_classes = 5
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the Deep Neural Network model with L2 regularization and dropout
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu', kernel_re
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.0
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.0
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))  # For multi-class classi

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac

# Fit the model on the training data
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test

# Evaluate the model on the test data
_, accuracy = model.evaluate(X_test, y_test)
print(f"Accuracy: {accuracy}")
```

```
Epoch 1/10
9962/9962 [==============================] - 17s 2ms/step - loss: 1.3797 - a
ccuracy: 0.4311 - val_loss: 1.3037 - val_accuracy: 0.4449
Epoch 2/10
9962/9962 [==============================] - 15s 1ms/step - loss: 1.3145 - a
ccuracy: 0.4410 - val_loss: 1.2915 - val_accuracy: 0.4472
Epoch 3/10
9962/9962 [==============================] - 14s 1ms/step - loss: 1.3073 - a
ccuracy: 0.4420 - val_loss: 1.2914 - val_accuracy: 0.4447
Epoch 4/10
9962/9962 [==============================] - 15s 2ms/step - loss: 1.3043 - a
ccuracy: 0.4423 - val_loss: 1.2825 - val_accuracy: 0.4493
Epoch 5/10
9962/9962 [==============================] - 15s 1ms/step - loss: 1.3024 - a
ccuracy: 0.4427 - val_loss: 1.2853 - val_accuracy: 0.4477
Epoch 6/10
9962/9962 [==============================] - 15s 1ms/step - loss: 1.3020 - a
ccuracy: 0.4422 - val_loss: 1.2852 - val_accuracy: 0.4466
Epoch 7/10
9962/9962 [==============================] - 15s 1ms/step - loss: 1.3003 - a
ccuracy: 0.4429 - val_loss: 1.2781 - val_accuracy: 0.4480
Epoch 8/10
9962/9962 [==============================] - 14s 1ms/step - loss: 1.2996 - a
ccuracy: 0.4425 - val_loss: 1.2776 - val_accuracy: 0.4485
Epoch 9/10
9962/9962 [==============================] - 14s 1ms/step - loss: 1.3002 - a
ccuracy: 0.4424 - val_loss: 1.2863 - val_accuracy: 0.4447
Epoch 10/10
9962/9962 [==============================] - 15s 1ms/step - loss: 1.2993 - a
ccuracy: 0.4429 - val_loss: 1.2788 - val_accuracy: 0.4480
2491/2491 [==============================] - 2s 973us/step - loss: 1.2788 -
accuracy: 0.4480
Accuracy: 0.44802749156951904
```

Gaussian Naive Bayes

In [12]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Assuming 'fcd_df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = fcd_df.drop('Type_encoded', axis=1)  # Features
y = fcd_df['Type_encoded']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Initialize the MinMaxScaler and fit-transform the training data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)

# Transform the test data using the same scaler
X_test = scaler.transform(X_test)

# Initialize the Gaussian Naive Bayes classifier
clf = GaussianNB()

# Fit the classifier on the training data
clf.fit(X_train, y_train)

# Predict on the test data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.41406093153985146
```

KNN-Model

In [13]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score



# Features (X) and target variable (y)
X = fcd_df.drop('Type_encoded', axis=1)  # Features
y = fcd_df['Type_encoded']  # Target variable


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the k-NN classifier
k = 5  # You can change this value as needed
knn = KNeighborsClassifier(n_neighbors=k)

# Fit the model on the training data
knn.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = knn.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the k-NN model: {accuracy:.2f}')
```

```
Accuracy of the k-NN model: 0.86
```

Best Model Performance on unseen data(New Simulation data)

In [14]:
```
fcd_df_New = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/fcd-ou
```

In [15]:
```
fcd_df_New['id'] = fcd_df_New['id'].str.replace(r'^f_', '', regex=True)
```

In [16]:
```
from sklearn.preprocessing import LabelEncoder
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns
fcd_df_New['Type_encoded'] = label_encoder.fit_transform(fcd_df_New['type'])
fcd_df_New['lane'] = label_encoder.fit_transform(fcd_df_New['lane'])
```

In [17]:
```
fcd_df_New.drop(columns=['type'],inplace=True)
fcd_df_New.drop(columns=['id'],inplace=True)
```

In [18]:
```
import pandas as pd

# Assuming df is your DataFrame
# Shuffle the rows using sample() function
shuffled_df = fcd_df_New.sample(frac=1, random_state=42)  # frac=1 shuffles tl

# Reset the index if needed
shuffled_df.reset_index(drop=True, inplace=True)
```

In [19]:
```
shuffled_df
```

Out[19]:

	time	x	y	angle	speed	pos	lane	slope	acceleration	Type_encoded
0	50.3	1.65	-1.6	90.0	7.35	9.52	46	0	0.93	3
1	587.9	-14.82	-1.6	90.0	9.65	85.18	35	0	0.90	3
2	440.3	24.21	4.8	270.0	14.03	58.54	9	0	0.91	1
3	834.9	58.19	-4.8	90.0	13.61	66.06	45	0	0.69	2
4	196.8	81.08	4.8	270.0	21.35	1.67	9	0	0.53	3
...	...	...	...	...	...	...	...	...	...	...
398471	651.7	108.07	-1.6	90.0	21.03	16.17	38	0	2.95	4
398472	918.4	312.23	-4.8	90.0	18.98	5.85	39	0	0.28	3
398473	333.6	52.71	-1.6	90.0	14.44	60.58	46	0	0.85	3
398474	370.9	139.56	-4.8	90.0	17.69	47.66	37	0	0.40	2
398475	308.6	191.37	8.0	270.0	13.13	22.70	2	0	0.56	3

398476 rows × 10 columns

Random Forest Evaluation

In [22]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Assuming 'df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = shuffled_df.drop('Type_encoded', axis=1)  # Features
y = shuffled_df['Type_encoded']  # Target variable



# Predict on the test data
y_pred = clf_rand.predict(X)

# Calculate accuracy
accuracy = accuracy_score(y, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.5632560053804997
```

In [ ]: